1 Approach

We present a novel approach to PoW difficulty scaling. We measure average time between requests and scale problem difficulty individually. Our model:

```
func rp_scale_model(p Param) Difficulty {
        if math.Max(p.Cpu.Load, p.Cpu.Avg) < cpu_thres {
                return ZeroDifficulty
        }
        if p.Local.LongMean > 2*max(p.Global.ShortMean, p.Global.LongMean) {
                if p.Local.ShortMean > 3*p.Global.ShortMean
                         && math.Max(p.Cpu.Load, p.Cpu.Avg) < cpu_thres+20 {
                        return ZeroDifficulty
                }

                return BaseDifficulty
        }
        return *BaseDifficulty.multiply(1 + int((math.Max(p.Cpu.Avg, cpu_thres) - cpu_thres))).multip
}
```

Is compared to cpu_scaling:

```
func cpu_equal(p Param) Difficulty {
        if p.Cpu.Avg < cpu_thres-30 {
                return ZeroDifficulty
        }
        return *BaseDifficulty.multiply(1 + 4*int((math.Max(p.Cpu.Avg, cpu_thres)-cpu_thres)))
}

```

2 Results

2.1 Mitigation against Server Flooding

Table 0.1. Classical Proof of Work

	Attackers		Legitimate users		Mobile devices	
Protection type	CPU	Request rate	Response time	Solving time	Response time	Solving time
300	13.87	6.42	0.465	0.39	28	0
200	14.03	4.09	0.467	0.367	18	0
100	13.63	1.75	0.477	0.345	13	0
50	13.75	0.30	0.482	0.292	12	0

2.2 Mitigation against Server Draining